TRABAJO FIN DE GRADO

Implementación de filtros adaptativos sobre series temporales

Realizado por
Julen Beldarrain Portugal

Para la obtención del título de
Grado en Física

Director
Francisco Matorras Weinig
Co-Director
Carlos A. Meneses Agudo

Realizado en
C.I.C. Consulting Informático

Convocatoria de Junio, curso 2020/2021

Agradecimientos

Quiero agradecer a mi familia, a mi pareja y a mis amigos por ser un apoyo fundamental en estos años como estudiante.

Agradezco sinceramente a Francisco Matorras por su paciencia y dedicación a la hora de guiarme en este proyecto y siempre estar disponible ante cualquier duda.

Y por último, agradecer a la empresa C.I.C. consulting por brindarme la oportunidad de trabajar con ellos en este proyecto sobre todo a Carlos por estar siempre disponible cuando lo necesitaba y a mis compañeros de la empresa.

Resumen

En este proyecto se ha implementado el algoritmo de Kalman como función para el producto IDbox de CIC Consulting con la finalidad de obtener un filtro adaptativo capaz de ser implementado tanto a intervalos fijos de datos como a señales a tiempo real. Además, se han realizado diferentes pruebas con diferentes modelos dinámicos para mejorar el rendimiento de filtro dadas las características de la herramienta IDbox y se le han implementado algoritmos recursivos para mejorar la robustez del filtro.

Palabras clave: Filtro de Kalman, Filtro recursivo, seguimiento de señal, Filtrado a tiempo real, MATLAB

Abstract

Kalman algorithm has been implemented as a function for the CIC Consulting IDbox product in order to obtain an adaptative filter capable of being implemented both at fixed data intervals and at real-time signals. In addition, different tests have been carried out with different dynamic models to improve filter performance, given the characteristics of the IDbox tool, and recursive algorithms have been implemented to improve the robustness of the filter.

Keywords: Kalman Filter, Recursive Filter, Signal Tracking, Realtime Filtering, MATLAB

Índice general

Índice de figuras

Índice de extractos de código

1. Introducción

1.1. Descripción del problema

El producto IDbox de CIC Consulting es un software de monitorización de activos y, por ende, se dedica en su mayor parte a la ingesta, persistencia, representación y análisis de series temporales. Dentro del core del producto existe un motor de cálculo que se puede entender como una pieza software que permite operar con los datos entrantes [1]. El proyecto propuesto tiene como objetivo ampliar este motor de cálculo haciéndolo compatible con nuevas estructuras de datos y manejo de memoria para poder introducir filtros adaptativos (filtro de Kalman, Wiener, LMS, redes neuronales, etc) como una pieza fundamental del motor.

El objetivo principal es reducir el ruido de las señales recibidas y además intentar predecir dichas señales.

1.1.1. Requisitos de información

En primer lugar, se pensó que las señales que se tendrían que filtrar serian provenientes exclusivamente de la red eléctrica. Sin embargo, al final el alcance del filtro era para cualquier señal que entrara en la herramienta IDbox. Por tanto, no se tendría información sobre el origen del error de las señales ni que se desea limpiar. El único dato posible de obtener seria la frecuencia de muestreo en algunos casos no en todos.

1.1.2. Requisitos funcionales

Los requisitos funcionales que se requerían en IDbox eran los siguientes. Por un lado, que el filtro fuese capaz de funcionar de forma recursiva por la entrada de datos continua que se tiene en en la herramienta IDbox, también, un requisito fundamental para poder implementarlo a tiempo real era que el consumo de memoria no fuese muy elevado, es decir, que para el funcionamiento del filtro no se tuvieran que guardar grandes cantidades de datos.

Por otro lado, al ver que la predicción 4.2.1 funcionaba tan bien, también se contempló la idea de utilizar el filtro como "tracker" y rellenar con este los datos que pudieran faltar en algunas señales. En principio no era un requisito que se pedía para el filtro pero al final se decidió utilizar esta cualidad del filtro de Kalman.

1.1.3. Solución del problema

Debido a los requisitos tanto de información como funcionales el filtro por el que se optó para la solución del problema fue el filtro de Kalman, ya que, se ajusta mucho mejor a lo deseado para el filtrado. Además, con la implementación de la σ_q variable se hizo un algoritmo mucho mas robusto para implementarlo a la hora de no tener un modelo que describa la señal. La única pega o inconveniente, es que el usuario tenga un mínimo de conocimiento de la señal y sepa que procesado busca como hemos comentado en en la sección 4.4.1

1.2. Filtros digitales

El término estimador o filtro se usa comúnmente para referirse a un sistema que está diseñado para extraer información sobre una cantidad determinada de interés a partir de datos ruidosos. En general, la teoría de la estimación (filtrado) encuentra aplicaciones en muchos campos diversos: comunicaciones, radar, sonar, navegación, sismología, ingeniería biomédica y financiera ingeniería, entre otros.[2]

Los filtros digitales tienen por objetivo aislar, eliminar o dejar pasar un subconjunto de datos de una señal para obtener información de la misma. Se tienen diferentes tipos de filtros digitales dependiendo el método empleado para extraer esta información como los filtros de frecuencias, que limpian en base a las frecuencias de la señal. En este caso nos centraremos en los filtros adaptativos debido a su propiedad recursiva. No existe una solución única para el problema del filtrado adaptativo lineal. Más bien, tenemos un "kit de herramientas" representado por una variedad de algoritmos recursivos, cada uno de los cuales ofrece características deseables propias. El desafío al que se enfrenta el usuario del filtrado adaptativo es, en primer lugar, comprender las capacidades y limitaciones de varios algoritmos de filtrado adaptativo y, en segundo lugar, utilizar este conocimiento en la selección del algoritmo apropiado para la aplicación en cuestión. Básicamente, podemos identificar dos enfoques distintos para derivar algoritmos recursivos para el funcionamiento de filtros adaptativos lineales. [2]

1.3. Diferentes filtros adaptativos

Encontramos diferentes tipos de filtros adaptativos pero los más utilizados y conocidos son los de Wiener y el de Kalman. El diseño de un filtro de Wiener requiere información a priori sobre las estadísticas del datos a procesar. El filtro es óptimo solo cuando las características estadísticas de los datos de entrada coinciden con la información a priori en la que se basa el diseño del filtro. Sin embargo, cuando esta información no se conoce completamente, es posible que no sea posible diseñe el filtro Wiener o, de lo contrario, el diseño ya no será óptimo.[2]

Debido, a la demanda de información de la señal que requerían los filtros de Wiener este tipo de filtros se descartó en primer lugar y se decidió implementar el filtro de Kalman.

2. El filtro de Kalman

2.1. Introducción

El filtro de Kalman proporciona el estimador lineal con menor error cuadrático medio, combinando la evolución de la variable con observaciones recogidas. Es además un proceso recursivo, por lo que no es necesario almacenar la información del estado en todos los instantes. El filtro de Kalman se sitúa dentro de los problemas de filtrado, que quedan englobados en la teoría de procesos estocásticos. Dado un fenómeno físico, lo intentamos modelizar a partir de sus comportamientos, y lo haremos utilizando leyes físicas conocidas u observaciones recogidas, las cuales relacionaremos entre si para obtener una salida del sistema. Sin embargo, un sistema determinista no es suficiente para llevar a cabo este análisis. Las razones son diversas. Por un lado, un modelo matemático acaba por recoger solamente aquellas características dominantes, por lo que muchos efectos quedan fuera del modelo. Por otro lado, los sistemas dinámicos no quedan definidos simplemente por los efectos que recogemos, sino que también existen ruidos que no podemos modelar de forma determinista. Por ejemplo, en el lanzamiento de una pelota hay factores como la velocidad del viento que no somos capaces de controlar. Esto formará parte de la aleatoriedad que un sistema determinista no recoge.[3]

2.2. Algoritmo de Kalman

Es un algoritmo de procesado de datos óptimo recursivo. Óptimo porque minimiza un criterio determinado y porque incorpora toda la información que se le suministra para determinar el filtrado. Recursivo porque no precisa mantener los datos previos, lo que facilita su implementación en sistemas de procesado en tiempo real. Por último, algoritmo de procesado de datos, ya que es un filtro, pensado para sistemas discretos. El filtro de Kalman es el principal algoritmo para estimar sistemas dinámicos representados en la forma de espacio-estado ya que el sistema es descrito por un conjunto de variables denominadas de estado. El estado contiene toda la información relativa al sistema a un cierto punto en el tiempo. Esta información debe permitir la inferencia del comportamiento pasado del sistema, presente o futuro, dependiendo si la problemática a encarar por parte del filtro de Kalman es el alisado, el filtrado o la predicción respectivamente. El objetivo del filtro de Kalman es estimar los estados de una manera óptima, de manera que se minimiza el índice del error cuadrático medio.[3]

Podemos definir el filtro de Kalman en dos procesos. Por un lado el de corrección (o observación) y por otro el de predicción. En la figura 2.1 podemos observar el algoritmo completo dividido en estas dos partes.

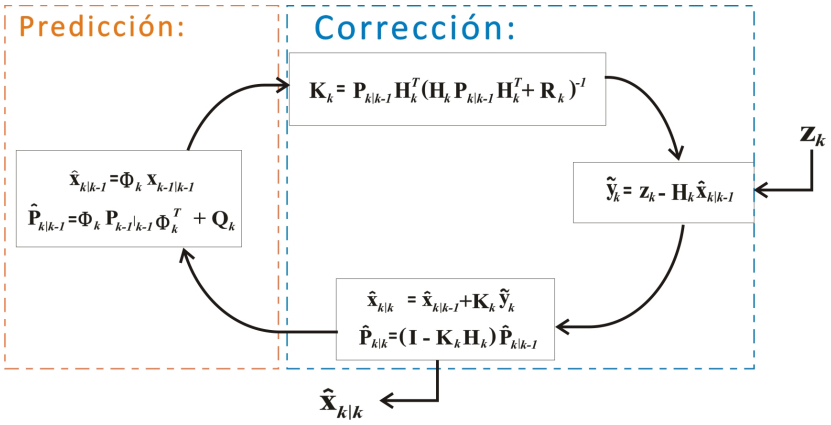


Figura 2.1: Esquema del algoritmo de Kalman.

Las ecuaciones de predicción son:

$$\hat{x}_{k|k-1} = \Phi_k \hat{x}_{k-1|k-1} \tag{2.1}$$

$$P_{k|k-1} = \Phi_k \hat{x}_{k-1|k-1}\Phi'_k + Q_k \tag{2.2}$$

Donde Φ_k es la matriz de transición del modelo dinámico en el instante k, $\hat{x}_{k|k-1}$ es la estimación del vector de estado en el instante k teniendo en cuenta la estimación del momento k-1, Para entenderlo mejor se puede pensar en el ejemplo del GPS, en este caso la posición del móvil estimada seria x_k y la trayectoria sin ruido estaría definida por la matriz Φ. $P_{k|k-1}$ es la matriz de covarianza del error, la cual nos dice como de bien sabemos dónde se encuentra el móvil y Q es la matriz del error de la covarianza, este nos da el error que cometemos en la estimación. En la parte de predicción el algoritmo se encarga de estimar el vector de estado para el instante k+1 tomando como referencia el estado en el momento n y de la actualización intermedia de la matriz de covarianza del estado. Y por otro lado las ecuaciones de corrección son:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \cdot (z_k - H_k \hat{x}_{k|k-1}) \tag{2.3}$$

$$P_{k|k} = \Phi_k \hat{x}_{k-1|k-1}\Phi'_k + Q_k \tag{2.4}$$

Donde K_k es la ganancia de Kalman que se define como

$$K_k = P_{k|k-1}H'_k(H_k P_{k|k-1}H'_k + R_k)^{-1} \tag{2.5}$$

Donde R es el error que tiene el aparato que estamos utilizando para medir. Por último z_k es el vector de las observaciones, la posición que medimos del movil, que lo podemos definir como:

$$z_k = Hx_{tk} + v_k \tag{2.6}$$

Donde H es el modelo de observación, nos permite decirle al filtro si estamos midiendo solamente la posición o también la velocidad, aceleración...del móvil, y v_k

es el ruido que se tendría en la medición. Estas últimas ecuaciones se encargan de incorporar nueva información con la llegada de la nueva observación y así mejorar la estimación teniendo en cuenta las anteriores estimaciones.

Destacar que la implementación práctica del filtro de Kalman es a menudo difícil debido a la dificultad de obtener una buena estimación de las matrices de covarianza de ruido Q y R

2.3. Optimidad y rendimiento

De la teoría se deduce que el filtro de Kalman es el filtro lineal óptimo en los casos en que:

- El modelo coincide perfectamente con el sistema real
- El ruido de entrada es blanco (no correlacionado)
- Las covarianzas del ruido se conocen con exactitud

Los ruidos correlacionados se pueden tratar explícitamente dentro del marco del filtro de Kalman. Se han propuesto varios métodos para la estimación de la covarianza del ruido durante las últimas décadas, incluido el ALS. Una vez estimadas las covarianzas, es útil evaluar el rendimiento del filtro; es decir, si es posible mejorar la calidad de la estimación del estado. Si el filtro de Kalman funciona de manera óptima, la secuencia de innovación (el error de predicción de salida) es un ruido blanco, por lo tanto, la propiedad de blancura de las innovaciones mide el rendimiento del filtro. Se pueden utilizar varios métodos diferentes para este propósito. Si los términos de ruido tienen una distribución no gaussiana, en la literatura se conocen métodos para evaluar el rendimiento de la estimación del filtro, que utilizan desigualdades de probabilidad o teoría de muestras grandes.[4]

2.4. MAD

El error absoluto medio o en inglés median absolute deviation(MAD), proporciona una medición del error promedio del pronóstico (en valor absoluto) y se define de la siguiente manera

$$MAD = mediana(x_i - media(x)) \tag{2.7}$$

Además, si las medidas son más o menos gaussianas se puede aproximar como $MAD \approx 0{,}67449\sigma$. Este método es un buen estimador para medir la dispersión en presencia de outliers, por lo que es interesante utilizarlo para contar número de puntos que se desvían, por ejemplo en un intervalo $2MAD$.

2.5. Suavizador RTS

El suavizador de intervalo fijo óptimo proporciona la estimación óptima utilizando las medidas de un intervalo fijo. Esto también se denomina "Suavizado de Kalman". Hay varios algoritmos de suavizado de uso común. En nuestro caso optamos por el Rauch-TungStriebel(RTS). [4]

El uso de este suavizador se separa en dos partes la primera sería la misma que el filtro de Kalman solo que esta vez, tendremos que guardar todas las estimaciones que ha hecho el filtro. Así, en la segunda parte se hace una pasada de la señal de atrás hacia adelante calculando así estimaciones de estado suavizadas. El algoritmo de la segunda parte sería el siguiente:

$$\hat{x}_{k|n} = \hat{x}_{k|k} + C_k(\hat{x}_{k+1|n} - \hat{x}_{k+1|k}) \tag{2.8}$$

$$P_{k|n} = P_{k|k} + C_k(P_{k+1|n} - P_{k+1|k})C'_k \tag{2.9}$$

Donde

$$C_k = P_{k|k}\Phi'_{k+1}P^{-1}_{k+1|k} \tag{2.10}$$

La contra parte de este método es que se tiene que trabajar con intervalos de datos por lo que no es lo más óptimo para utilizar a tiempo real cuando la entrada de datos es continua. Aun así, se puede implementar a la hora de aplicar el filtro a trenes de datos fijos a los que se les quiere hacer un análisis más fino.

2.6. Extensiones del filtro de Kalman

Hay otros métodos de implementación del filtro de Kalman pero estas ya tienen en cuenta que las observaciones afectan directamente al estado de la variable que se quiere medir. Estas aplicaciones son muy interesantes pero no se les vio aplicación en este proyecto. Aun así, para una lectura más allá de este proyecto las extensiones más conocidas son las siguientes

- Filtro de Kalman extendido (extended Kalman filter)
- Unscented Kalman filter
- Filtro de partículas

3. Aplicación del método de Kalman

Como se ha mencionado en los capítulos anteriores debido al problema que nos presentaron en el C.I.C. Consulting nuestra propuesta como solución fue la de implementar el filtro de Kalman. Para ello, a pesar de que el filtro final debería estar programado en C#, primero se intentó conseguir un filtro totalmente funcional en MATLAB, ya que es una herramienta más fácil de usar a la hora de manejar matrices y reducía la dificultad de programar para poder entender bien el filtro. y nos permitía realizar pruebas de los métodos sobre datos simulados en los que teníamos la referencia real.

3.1. Tiro Parabólico ruidoso

Para la implementación y prueba de las ecuaciones del filtro de Kalman se utilizó el siguiente sistema que representa un tiro parabólico sencillo que se genera de la siguiente manera.

$$x_{tk} = \Phi x_{tk-1} + Gg \tag{3.1}$$

Donde,

$$\Phi = \begin{pmatrix} 1 & dt \\ 0 & 1 \end{pmatrix} \tag{3.2}$$

y derivando:

$$G = \begin{pmatrix} -\frac{dt^2}{2} \\ -dt \end{pmatrix} \tag{3.3}$$

dt el paso de tiempo y x_{tk} el vector de estado en la momento k. El modelo de medida viene dado por:

$$z_k = Hx_{tk} + v_k \tag{3.4}$$

Donde,

$$H = (1\ 0) \tag{3.5}$$

es decir, solo medimos la posición del tiro. z_k es la posición en el instante k a la cual se le ha añadido un error aleatorio con varianza $R = 4$. Para inicializar el filtro tomamos estos valores iniciales para la matriz de covarianza:

$$P = \begin{pmatrix} 50 & 0 \\ 0 & 0{,}05 \end{pmatrix} \tag{3.6}$$

Asumimos que no hay ruido, por tanto,

$$Q = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \tag{3.7}$$

Y los valores iniciales del tiro parabólico serán $y_0 = 100, v_0 = 0$ .En la figura 3.1 se muestra el resultado de este modelo

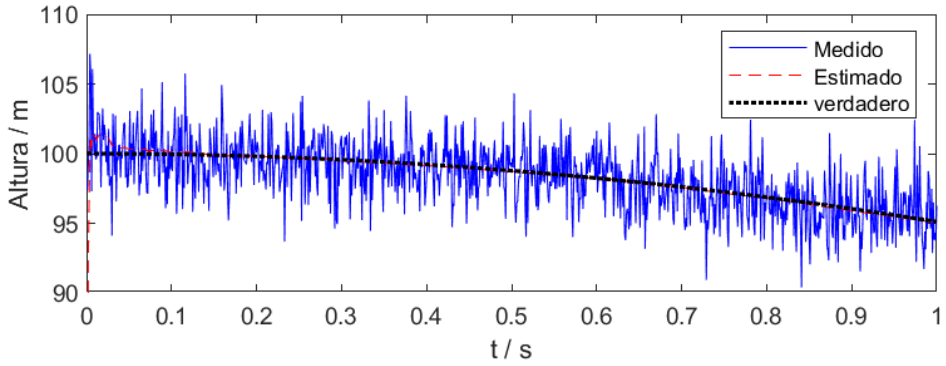


Figura 3.1: Ejemplo de aplicación del filtrado de kalman a un tiro parabolico ruidoso. La línea azul representa el valor medido del tiro , la roja representa el valor filtrado y la negra seria el valor verdadero del tiro.

Para la prueba del tiro, primero se generó la señal a partir de la ecuación de estado y después, se le introdujo ruido aleatorio gausiano

Para esta prueba se realizó una función que generaba una señal en base a unos parámetros ajustables, el tiempo, el paso, la desviación del ruido y una función (en este caso parabólica) que se genera de la ecuación de estado que utiliza el filtro. El código utilizado para recrear la aplicación del tiro se en cuentra en el anexo 7.4

3.2. Filtro KF 1D

Una vez se tuvo el algoritmo de Kalman bien implementado y probado decidimos utilizar una matriz de transición simple para aproximar las señales, por ello se optó por un modelo muy habitual en la aplicación del filtro de Kalman donde se supone velocidad constante y aceleración aleatoria, es decir, lo que sería un desarrollo de Taylor de primer orden, y de ahí surge el llamarlo modelo 1D haciendo referencia al orden del desarrollo. Por tanto, el modelo dinámico sería de la siguiente manera. El vector de estado se define como:

$$x_t = (x_t \ v_t)^T \tag{3.8}$$

Donde x_t y v_t son los valores verdaderos de la posición y la velocidad. Para este modelo estamos teniendo en cuenta que la velocidad es constante entre cada medida por lo que la matriz de transición viene dado por :

$$\Phi = \begin{pmatrix} 1 & T \\ 0 & 1 \end{pmatrix} \tag{3.9}$$

Donde T es el paso de tiempo. Por otro lado asumimos que el error viene dado por la aceleración y que esta es aleatoria, y escribimos por tanto la matriz del error de

la covarianza como:

$$Q = \begin{pmatrix} T^4/4 & T^3/2 \\ T^3/2 & T^2 \end{pmatrix} \sigma_q^2 \tag{3.10}$$

Donde σ_q^2 es la varianza del error. Destacar que la elección de este parámetro es muy delicado ya que determina directamente junto a la varianza del error de la medida R el desempeño del filtrado. En el apartado 4.2 se discutirá como afecta la elección de estos dos parámetros[5] . En la figura 3.2 se muestra el resultado de este modelo

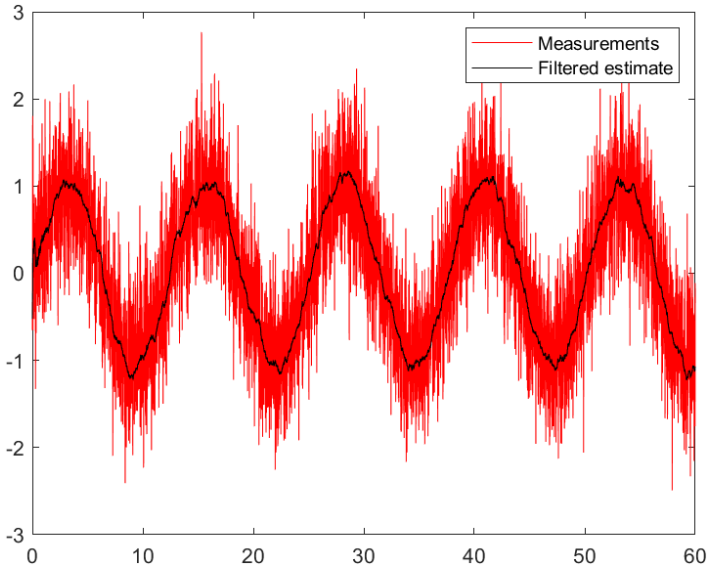


Figura 3.2: Ejemplo de uso del modelo 1D para filtrar un seno ruidoso. En rojo se muestra el valor de la medida y en negro el valor del filtrado de Kalman

El proceso de prueba de este modelo fue parecido al del tiro parabólico, el único cambio fue que, ahora, la señal verdadera no se obtenía en base a la ecuación de estado sino que, se generó con una función matemática, se puede observar el procedimiento de implementación y prueba en el código del anexo 7.4.

3.3. Validation Gate

Una vez implementado el filtro de Kalman 1D y se hicieron las pruebas nos dimos cuenta de que el filtrado era muy susceptible a los posibles outliers de la señal. Para ello, la solución planteada fue añadirle al algoritmo de kalman el llamado validation gate para poder identificar e ignorar en el proceso del filtrado valores anómalos. Lo que se busca con esta puerta es que, teniendo en cuenta la covarianza de la medida anterior observar si el valor que entra, la nueva observación, se desvía más de lo que se esperaría o de lo que se desearía y en este caso denegar la entrada de este dato al filtro y predecir el siguiente punto con los valores anteriores. Para ello recordemos la

predicción de la covarianza de la medida

$$S_{k+1} = H_{k+1} P_{k-1,k} H'_{k+1} + R_{k+1}$$

La predicción de la medida

$$\hat{z}_{k+1,k} = H_k \hat{x}_{k+1,k}$$

Y el residuo de la medida

$$v_{k+1} = z_{k+1} - \hat{z}_{k+1,k}$$

Podemos construir un validation gate entorno a la medida de la forma

$$e^2 = v_{k+1} S_{k+1}^{-1} v'_{k+1} \leq \chi^2 \tag{3.11}$$

El error e^2 varía como una distribución Chi-cuadrado con el número de medidas como grados de libertad se tengan, por lo que, se puede elegir un valor de χ^2 para tener un grado de confianza deseado.

A la hora de implementar el algoritmo en forma de función se separó el algoritmo en dos funciones para una mejor comprensión del filtrado. Los códigos de las dos funciones se pueden observar en el anexo, por un lado el de predicción 7.1 y por otro lado el de corrección 7.2, en este último fue donde se implementó el validation gate.

3.4. Filtro KF 2D

Una vez se tuvo el validation gate implementado se puede observar en la sección 4.2.1 que debido a la ecuación 3.2 la predicción es una recta con lo que el filtrado no es del todo satisfactoria a pesar de reducir el RMS. Por ello, desarrollamos la matriz de transición.

Como en la sección 3.2 se suponía una función de la cual se conocía hasta su primera derivada la idea que se tuvo fue la de desarrollar un orden más el polinomio de Taylor para mejorar en cuanto a suavidad del modelo, sobre todo, en puntos donde la primera derivada se hace 0 y el modelo 1D no es muy consistente. El desarrollo sería de la siguiente manera

$$x = x_0 + \frac{dx}{dt}t + \frac{1}{2}\frac{d^2x}{dt^2}t^2 + \frac{1}{6}\frac{d^3x}{dt^3}t^3 \tag{3.12}$$

$$v = v_0 + \frac{d^2x}{d^2}t + \frac{1}{2}\frac{d^3x}{dt^3}t^2 \tag{3.13}$$

$$a = a_0 + \frac{d^2x}{dt^2} + \frac{d^3x}{dt^3}t \tag{3.14}$$

En este nuevo modelo suponemos que conocemos la función hasta su segunda derivada y que el error es del orden de la tercera derivada, por tanto, nuestra nueva

matriz de transición será la siguiente.

$$\Phi = \begin{pmatrix} 1 & T & T^2/2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{pmatrix} \tag{3.15}$$

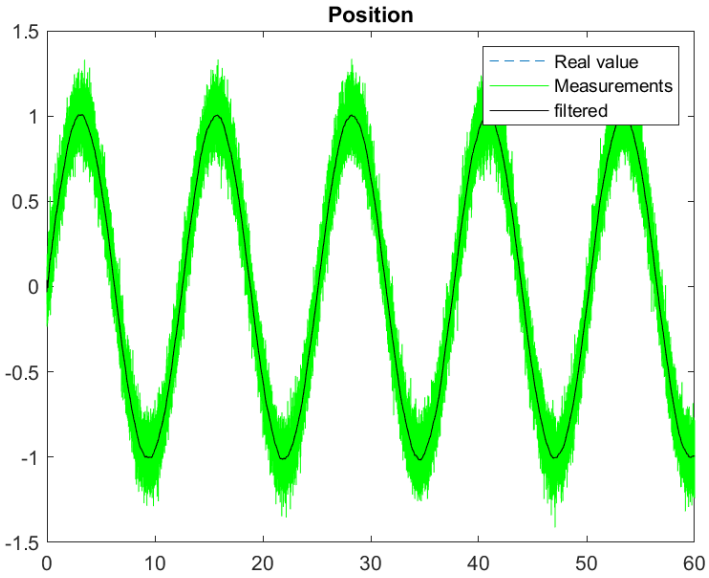


Figura 3.3: Filtrado de un seno ruidoso con el modelo 2D.

Se observa que con una buena elección de los parámetros el filtrado puede llegar a ser más suave que el anterior modelo dinámico. La forma de generar la señal es la misma que con el anterior modelo, pero en este caso, se tenía que tener en cuenta el cambio de dimensiones tanto en la matriz H ,R y P como en los vectores del valor estimado, debido que al aumentar la dimensión del modelo, el filtro devuelve estimaciones de la posición, velocidad y aceleración . Se puede ver la implementación en el código del anexo 7.5

3.5. Matriz Q

Como ya se ha mencionado anteriormente, la elección de la matriz Q es bastante delicada, pues implica que conoces la varianza del error de tu señal σ_q^2. En nuestro caso este valor era imposible de conocer debido a la gran diversidad de señales que se manejan en IDbox. Por tanto, utilizamos una matriz Q consistente al modelo que estamos utilizando.

Ahora la matriz Q la construimos en base a la última derivada. Multiplicamos cruzadamente y así obtenemos la matriz de covarianza despreciando términos de orden mayor a 3, obteniendo la siguiente matriz del error de la covarianza.

$$Q = \begin{pmatrix} T^6/36 & T^5/12 & T^4/6 \\ T^5/12 & T^4/4 & T^3/2 \\ T^4/6 & T^3/2 & T^2 \end{pmatrix} \sigma_q^2 \tag{3.16}$$

A parte de esta matriz Q, encontramos en diferentes artículos[6] [7] y paginas web el uso de la siguiente matriz

$$Q = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & T^2 \end{pmatrix} \sigma_q^2 \tag{3.17}$$

Aun así, no hemos encontrado una justificación sólida (más allá de su simplicidad) y se comprobó que su rendimiento sobre nuestros ejemplos es peor. Una razón por la que se puede utilizar es que, como el coeficiente de mayor error es 3.14 solamente tienes en cuenta este y desprecias todos los demás. sin embargo, se optó por implementar 3.16 debido a que era más consistente y razonable con el modelo dinámico que se utiliza.

3.5.1. σ_q variable

Además de la forma de la Q, es crítico la elección de sigma, que debe tener un valor del orden de la derivada segunda (1D) o tercera (2D) de los datos. La importancia de elegir bien el orden de magnitud se discute en detalle en 4.4 . Por lo que se optó por aproximar el orden de magnitud de σ_q para que el filtrado fuera más robusto. La idea para aproximar el valor es la siguiente. Como el modelo que estamos utilizando asume que conocemos la función hasta su segunda derivada, es decir, posición, velocidad y aceleración y que la tercera derivada es aleatoria. Partimos de que el error viene de la tercera derivada por lo que se decidió intentar buscar el valor aproximado ajustando n puntos a un polinomio de 3^{o} grado y así se almacenaría el coeficiente de mayor orden, el que suponemos que es aleatorio, y intentamos obtener el orden de magnitud haciendo una media con los coeficientes que se vayan obteniendo en cada ajuste.

Para la implementación en IDbox se pensó en una manera de que el algoritmo fuese reajustando el parámetro cada n puntos. La idea sería la siguiente. Partir de un ajuste a un polinomio n puntos (por comodidad coger un número impar 2n+1 e ir de -n a n), en el caso práctico se utiliza un polinomio de orden tres pero de forma general quedaría como

$$\sum_{j=0}^{m} x^j a_j \tag{3.18}$$

en este caso recordar que el que interesa es a_3. Ahora, si suponemos que todos los puntos tienen el mismo error, hacemos mínimos cuadrados, o sea calcular los coeficientes a que hacen mínimo

$$\sum_{i=0}^{n} [y_i - \sum_{j=0}^{m} x_i^j a_j]^2 \tag{3.19}$$

Se busca el mínimo, haciendo cero la derivada con respecto a cada parámetro a_k

$$\sum_{i=0}^{n} y_i \cdot x_i^k = \sum_{i=0}^{n} [\sum_{i=0}^{m} x_i^j a_j]_i^k \text{ para } k = 0, 1, 2, 3 \tag{3.20}$$

Organizando sumatorios y poniéndolo en forma vectorial se tiene

$$B\vec{a} = \vec{u} \;\; \vec{a} = B^{-1}\vec{u} \tag{3.21}$$

con

$$u_k = \sum_{i=0}^{n} y_i \cdot x_i^k \tag{3.22}$$

y

$$B_{kj} = \sum_{i=0}^{n} [x_i^{j+k}] \tag{3.23}$$

Una cosa interesante es que, si vas deslizando la ventana de ajuste, los x son siempre los mismos (una vez que has fijado n), puedes calcular B y la inversa antes de empezar a ver los datos. No tienes que almacenar nada para el cálculo de u se optó por calcularlo a partir del valor anterior, añadiendo el nuevo y quitando el más antiguo. Esto solo supone tener que almacenar las n medidas y no toda la señal. Destacar que esta es una característica esencial para el filtrado a tiempo real. La implementación del algoritmo de la σ_q variable se muestra en el código del anexo 7.6

El único problema de este método de aproximación es la elección del número de puntos que se elige para el ajuste, se comentara más en detalle en el próximo capítulo como afecta este parámetro por lo que la mejor opción cara a la implementación en la herramienta IDbox es la de dejarlo a elección del usuario.

4. Aplicación del filtro de Kalman

4.1. Introducción

Después de describir en el anterior capítulo como fue la implementación y los pasos seguidos, en este capítulo se mostraran varios resultados obtenidos con los diferentes métodos.

4.2. Pruebas en MATLAB

A continuación se muestra la aplicación del filtro en sus condiciones óptimas.

En la Figura 3.1 podemos observar en azul los valores generados de la ecuación de estado del tiro parabólico tras añadirle un ruido aleatorio, en negro el valor verdadero, es decir, los valores que realmente nos interesan y también los valores filtrados en rojo. Se observa también que el filtro tarda un poco en converger al valor verdadero pero que una vez converge limpia muy bien la señal. Además el filtro de Kalman nos devuelve valores de la velocidad del tiro parabólico.

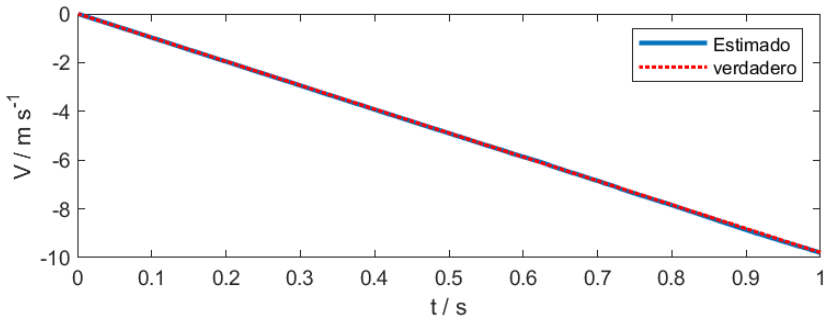


Figura 4.1: Valor verdadero de la velocidad del tiro y los valores del filtro en función del tiempo.

Observando las figuras 3.1 y 4.1 se puede comprobar que el filtro está bien implementado las ecuaciones eran correctas como, a parte de recuperar el valor verdadero del tiro parabólico, es capaz de proporcionar la velocidad de la señal (la derivada primera). Por tanto, para la siguiente fase de pruebas utilizamos el filtro sobre señales conocidas pero que no estaban generadas por la ecuación de estado que le pasábamos al filtro, es decir, la ecuación de estado era desconocida. A continuación se muestran señales de un seno ruidoso filtrados con el sistema dinámico 1D de la sección 3.2.

Para la prueba de la figura 3.2 se tomó un seno y se le añadió ruido aleatorio gaussiano con una desviación estándar de $\sigma = 0{,}5$ y un paso de tiempo de $T = 0{,}01$. Y para inicializar el filtro se tomaron los siguientes valores:

$$\hat{x}_0 = (y(0)\ 0)'$$

$$P = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$R = \sigma^2$$

$$\sigma_q = 0{,}025$$

$$H = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

Se puede observar en la figura 3.2 que el filtro consigue reducir el error y limpiar en gran medida la señal. A continuación muestra el histograma de los residuos de la señal filtrada

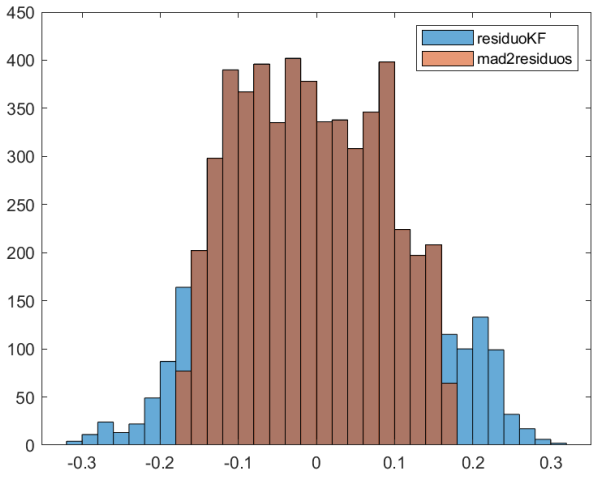


Figura 4.2: Histograma del seno ruidoso filtrado y en rojo los puntos que se encuntran dentro del intervalo 2MAD.

Analizando la figura 4.2 se pudo comprobar que el error se ha disminuido obteniendo que ahora la desviación de la señal filtrada es de $\sigma_{KF} = 0{,}08324$. Además, utilizando el MAD vemos que solo el 17 % se queda fuera del rango de $2 \cdot MAD \approx 1{,}5\sigma_{KF}$.
Otra señal:

En la figura 4.3 se aplica el filtro a un polinomio ruidoso. En este caso, como la señal es más suave el filtrado se observa que es bastante bueno y se reduce el error en gran medida reduciendo la desviación a un valor de $\sigma_{KF} = 0{,}06$. Y solo el 18 % de los puntos se desvían $2MAD \approx \sigma_{KF}$

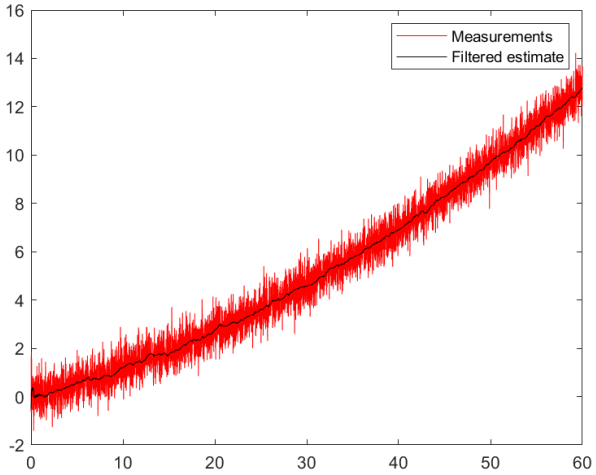


Figura 4.3: Filtrado de un polinomio ruidoso con el modelo 1D.

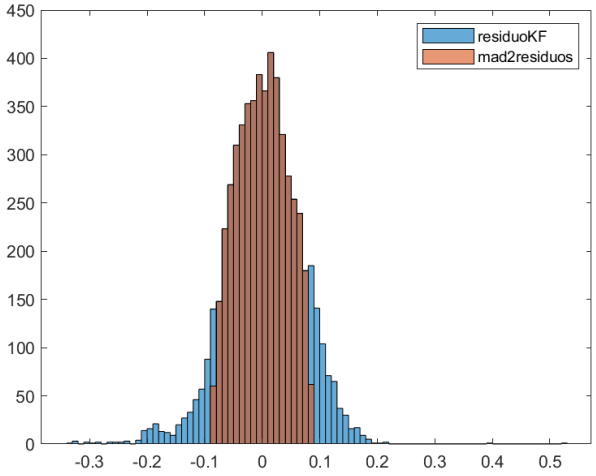
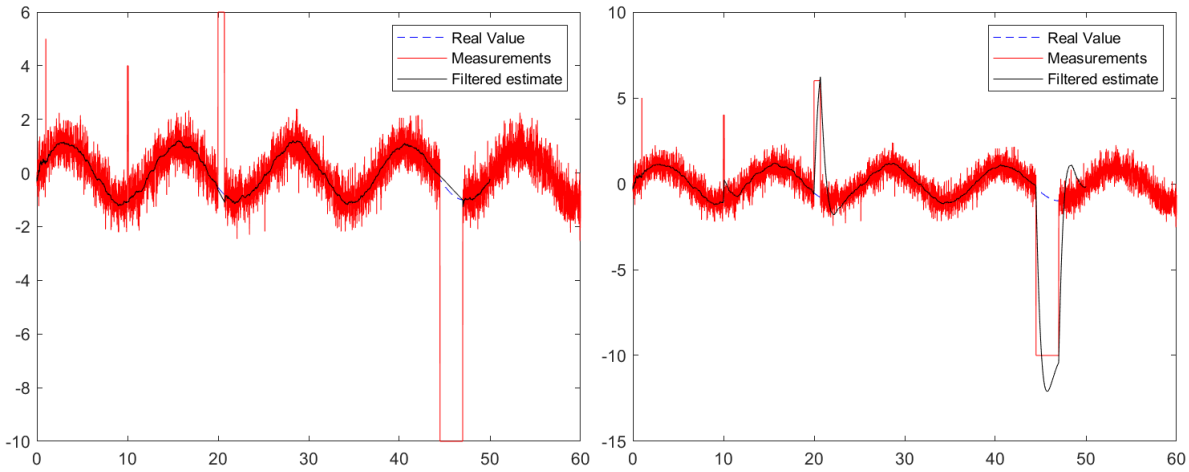


Figura 4.4: Residuo del filtrado del polinomio y valores en un rango 2MAD.

4.2.1. Respuesta frente valores anómalos

A continuación, se muestra el rendimiento del validation gate de la sección 3.3 en señales donde se han inducido outliers y además se ha simulado una pérdida de la señal con un valor para el validation gate de $\chi = 3\sigma$.



(a) Validation Gate activado (b) Validation Gate desactivado

Figura 4.5: Aplicación del validation gate a la señal sinusoidal ruidosa con presencia de outliers. A la izquierda el validation gate esta activo y a la derecha se hadesactivado

En la figura 4.5 a la izquierda, se puede observar que el filtrado de la señal es parecida a la de 3.2 pero en este caso, además se añaden una serie de valores anómalos y se eliminan una serie de datos, para simular una respuesta errónea de los sensores. Gracias al validation gate los outliers se han podido suprimir y poder trackear la señal con una buena precisión con una valor de la desviación de $\sigma_{KF} = 0{,}1$. En el caso 4.5 a la derecha, el filtro intenta seguir a la señal en todo momento y en los puntos donde se tienen outliers el modelo dinámico no consigue converger bien dando lugar a estos picos o valores desviados. En este caso si calculamos la desviación respecto a la señal verdadera nos dan valores absurdos como $\sigma_{KF} = 3$.

4.2.2. Pruebas 2D

A continuación se muestran las pruebas realizadas con el modelo 2D en las que se utilizaron los siguientes valores:

- T=0.005
- σ=0.1
- σ_q=$1/T^2$
- $H = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

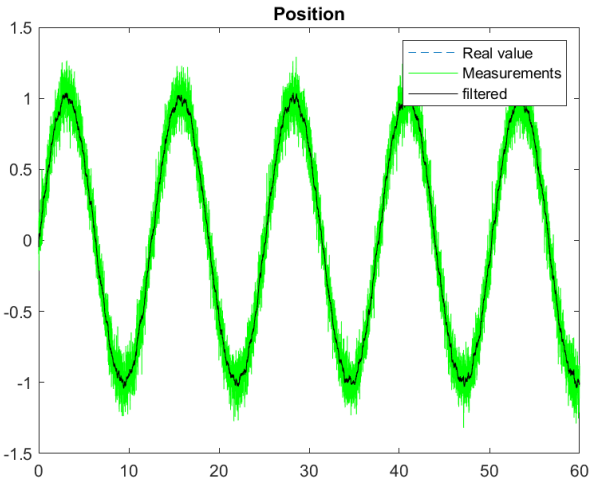


Figura 4.6: filtrado de la señal con el modelo 2D.

En la figura 4.6 el valor de la desviación filtrada es de $\sigma_{KF} = 0{,}02$. Y el 16 % de los puntos se desvían más de $2MAD$.

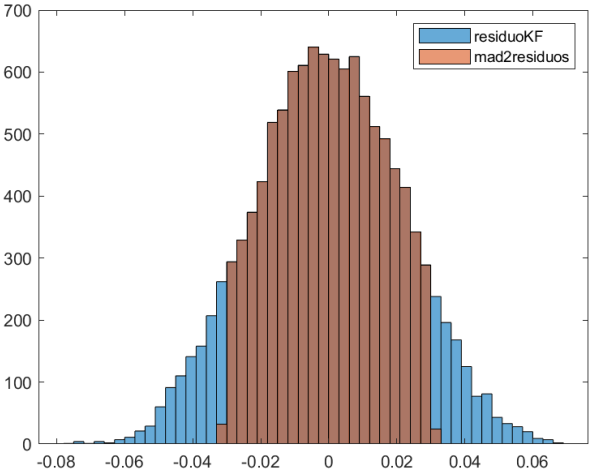


Figura 4.7: Residuo del filtrado del seno con el 2 D y valores en un rango de 2MAD.

En las figuras 4.8 y 4.9 se muestran los resultados de otra señal, sinusoidal pero con atenuación. Se puede observar que el resultado es peor, como es de esperar, cuanto mayor es el ruido con respecto a la señal real

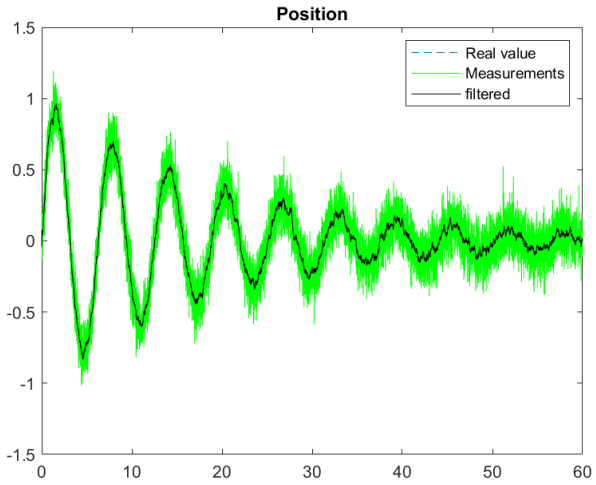


Figura 4.8: filtrado de la señal con el modelo 2D.

En la figura 4.8 el valor de la desviación filtrada es de $\sigma_{KF} = 0{,}0223$. Y el 16.9 % de los puntos se desvían más de $2MAD$.

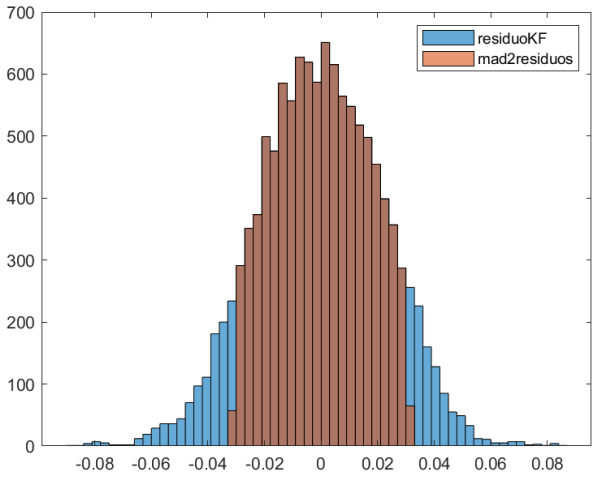


Figura 4.9: Residuo del filtrado del seno con el 2 D y valores en un rango de 2MAD.

4.2.3. Comparación 1D y 2D

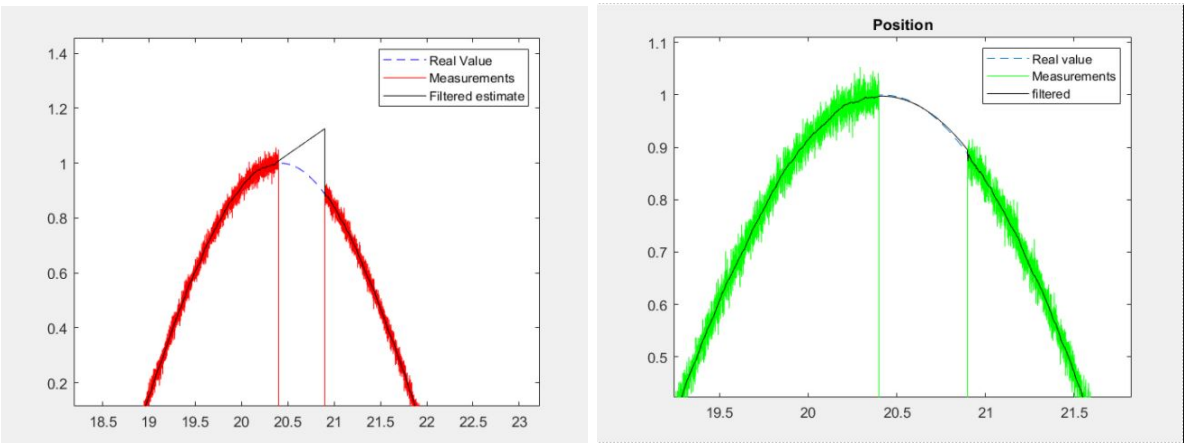
.

Como se ha podido comprobar en el anterior apartado el modelo dinámico 3.2 consigue reducir el error de la señal y trackear la señal en caso de pérdida de esta. Aun así, debido a la simplicidad del modelo los huecos los rellena con una recta, por ello, se decidió implementar el modelo dinámico 3.4 y así intentar predecir la señal con un nivel de suavizado. A continuación se mostrarán pruebas donde se utilizaron los dos modelos

Valores iniciales:

- $T = 0{,}0005$
- $\sigma = 0{,}002$
- $\sigma_q = 1/T^2$

En las figuras 4.10a y 4.10b se muestra la aplicación de los modelos 1D y el 2D en las zonas donde se tienen perdida de señal.



(a) Modelo 1D (b) Modelo 2D

Figura 4.10: Aplicación de los modelos 1D y 2D sobre señales donde se tiene perdida de señal. A la izquierda el modelo 1D y a la derecha el modelo 2D

Comparando en las figuras 4.10a y 4.10b podemos observar claramente como, debido al modelo 1D y la suposición de velocidad constante, el hueco se rellena con una recta y esto no es suficiente para seguir la curvatura de la señal, en cambio, al desarrollar un término más del polinomio de Taylor y tener en cuenta la segunda derivada (la aceleración) permite poder predecir la curvatura de la señal y seguirla mucho mejor. Por último destacar que en cuanto a la reducción del ruido sin presencia de outliers el modelo 2D no presentaba una mejora significativa, pero debido a como rellena los outliers y sobretodo los huecos o pérdidas de señal es una mejor opción para la implementación en la herramienta IDbox.

También, se forzó el filtro con señales con cambios abruptos para ver su respuesta. Para estas pruebas se utilizó una función salto y otra función tipo sierra (sawtooth).

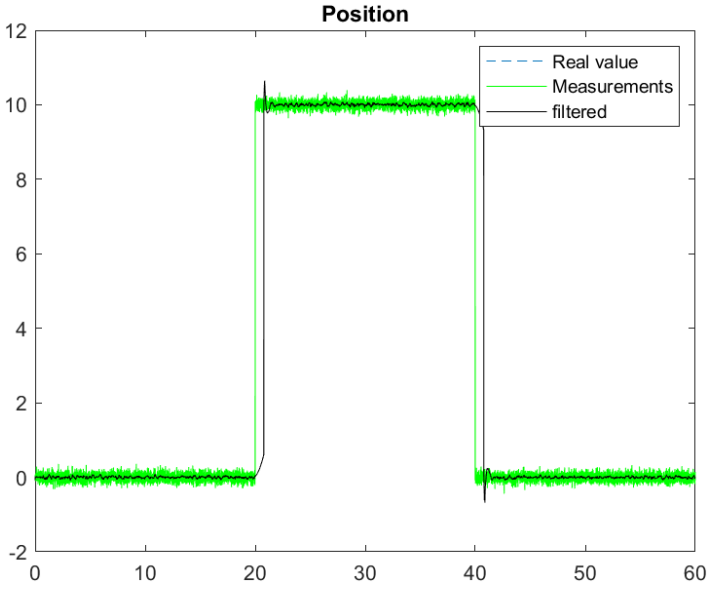


Figura 4.11: Filtrado de una función escalón.

y

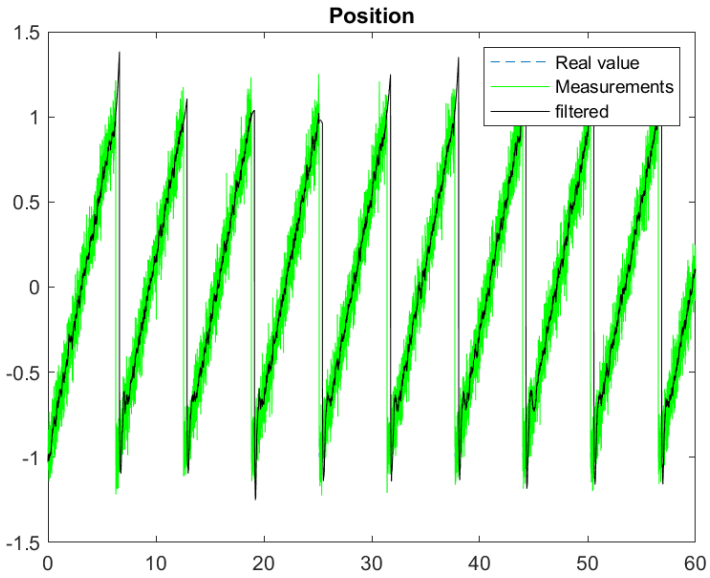


Figura 4.12: Filtrado de una función sierra.

Se puede observar en las figuras 4.11 y 4.12 que el filtro consigue seguir las dos funciones bastante bien.

4.3. Comparación Q

A continuación se muestran dos pruebas con las dos diferentes matrices Q que se comentaron en la sección 3.5.

En las figuras 4.13 y 4.14 se muestra un ejemplo de aplicación utilizando las matrices 3.16 y 3.17.

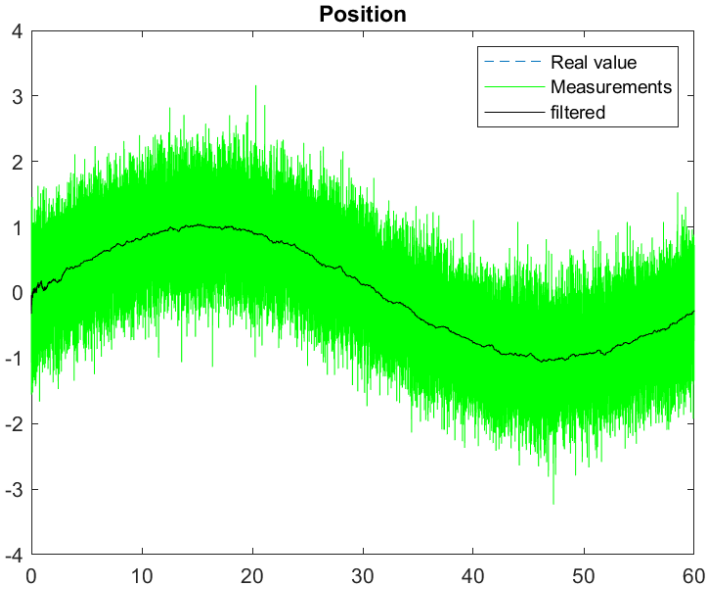


Figura 4.13: Ejemplo del filtrado de un seno ruidoso con la matriz Q 3.16

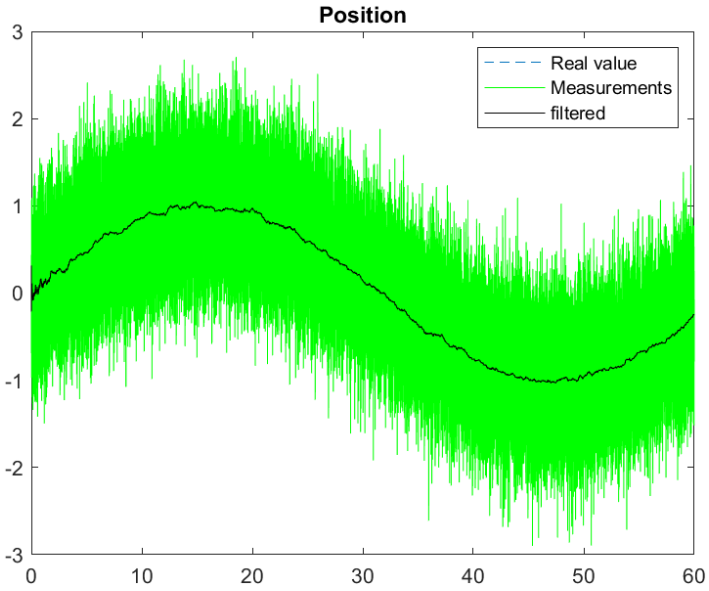


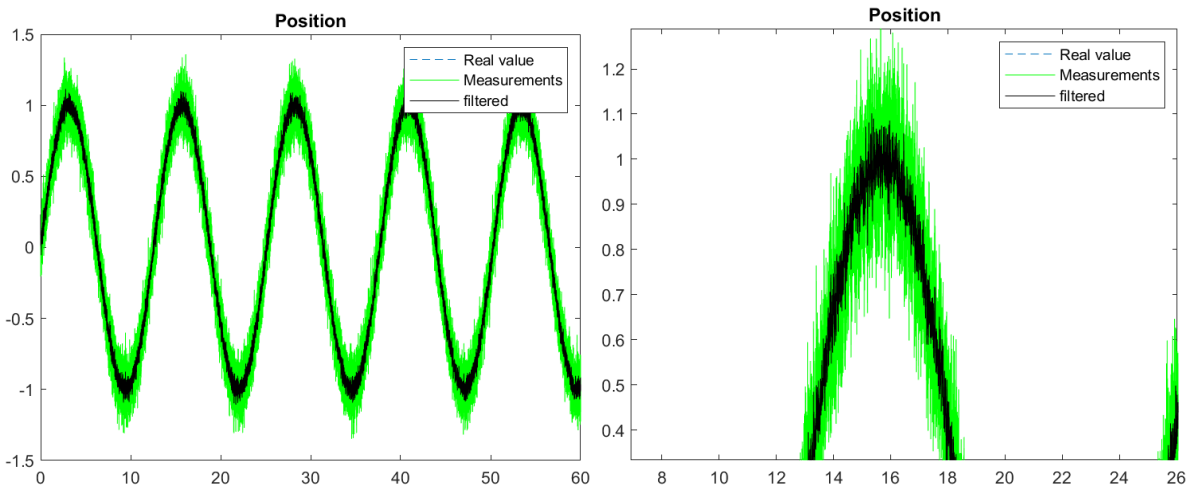
Figura 4.14: Ejemplo del filtrado de un seno ruidoso con la matriz Q 3.17

Se puede comprobar visualmente y también calculando la desviación, que los resultados son muy similares, comprobando que el uso de la matriz simplificada 3.17 es una buena aproximación de la real, al menos en los casos probados. A pesar de ello, se optó por la matriz completa, puesto que está justificada matemáticamente y no supone una complicación adicional ni de cálculo ni de implementación.

4.4. Problemas del filtrado

A pesar de que el filtro funcionara bien en los diferentes casos la elección de R y de σ_q es crucial. En el caso de R no es tan difícil de conseguir ya que viene asociado al aparato de medida que se esté utilizando. Lo único que se debe tener en cuenta es que, cuanto más aumentemos R más ruido se limpia, es decir, suaviza más la señal al darle más peso a la estimación previa del filtro, en cambio, si disminuimos R estamos diciéndole al filtro que le dé más peso a la medida y que siga más a la señal sin filtrarlo tanto. En cambio, el valor de σ_q implica conocer bien la dinámica del sistema y en el caso práctico de la herramienta IDbox, como lo que se busca es la aplicación en situaciones complejas, no hay una ley clara que nos pueda facilitar es información.

A continuación se muestran varios problemas en el filtrado debido a la elección de σ_q. En primer lugar, si cogemos un valor grande como se observa en 4.15 que el filtro sigue más a la señal y no lo filtra tan bien. Para las pruebas se introdujo ruido gausiano con $\sigma = 0{,}1$



(a) Filtrado de la señal (b) Zoom del filtrado

Figura 4.15: Ejemplo de aplicación del filtrado de la señal con un valor de σ_q grande.

Si analizamos el histograma del residuo de la figura 4.16 efectivamente vemos que en este caso la desviación es de $\sigma_{KF} = 0{,}04$.

En cambio, si reducíamos el sigma lo que se observaba era que el filtrado empezaba a ser cada vez más suave como se observa en la figura 4.17 y en este caso el filtrado era realmente bueno y mirando el histograma $\sigma_{KF} = 0{,}009$.

Pero, uno podría pensar que cuanto más pequeño se coja el valor mejor filtraría, pero no se puede abusar de esto ya que si el valor es excesivamente pequeño ocurre un sobre suavizado y se pierde por completo información de la señal como se observa en la figura 4.19.

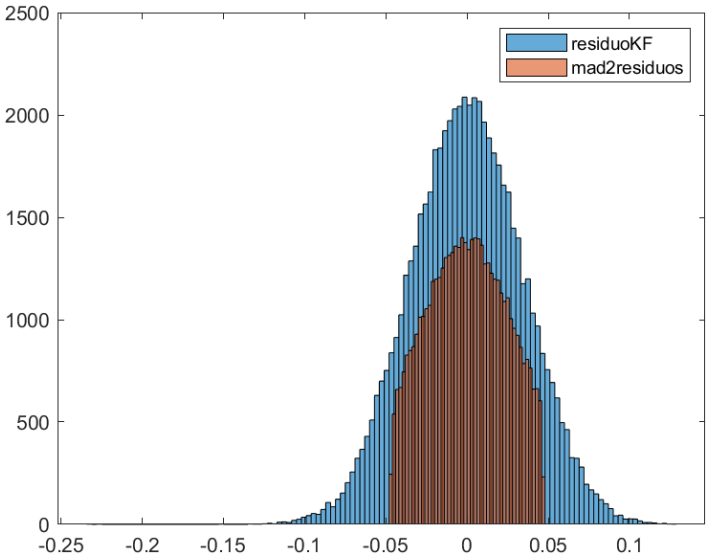


Figura 4.16: Histograma de la señal filtrada con σ_q grande.

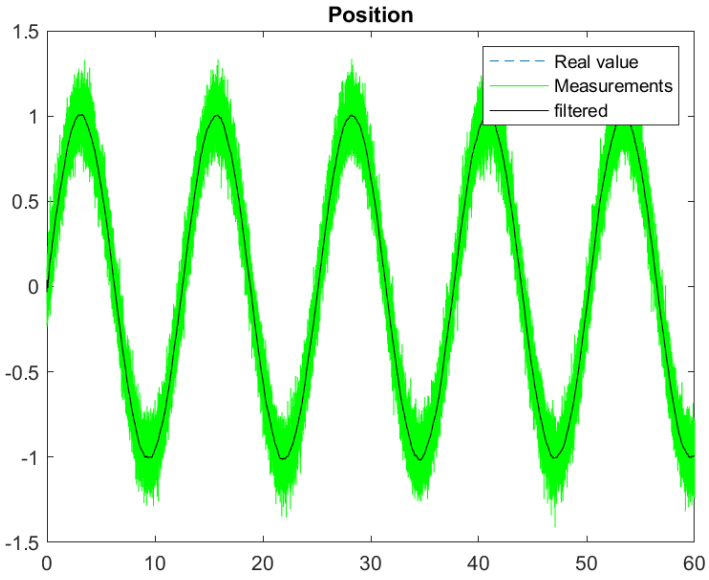


Figura 4.17: Filtrado de la señal con un valor de σ_q pequeño.

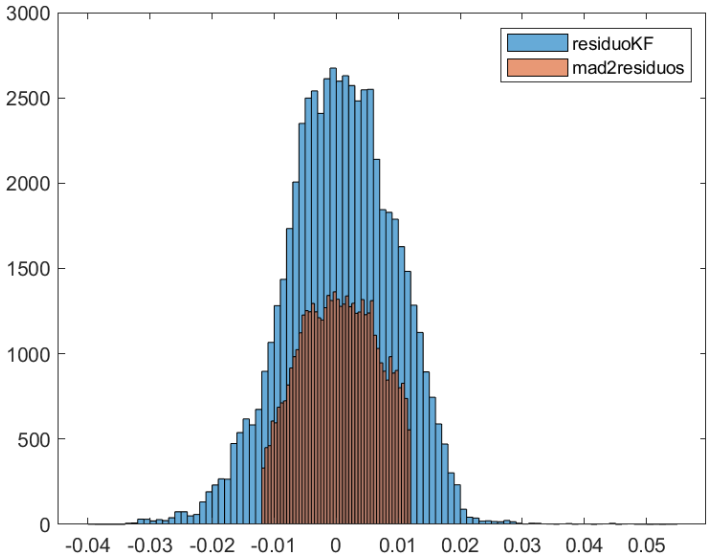


Figura 4.18: Histograma de la señal filtrada con σ_q pequeño.

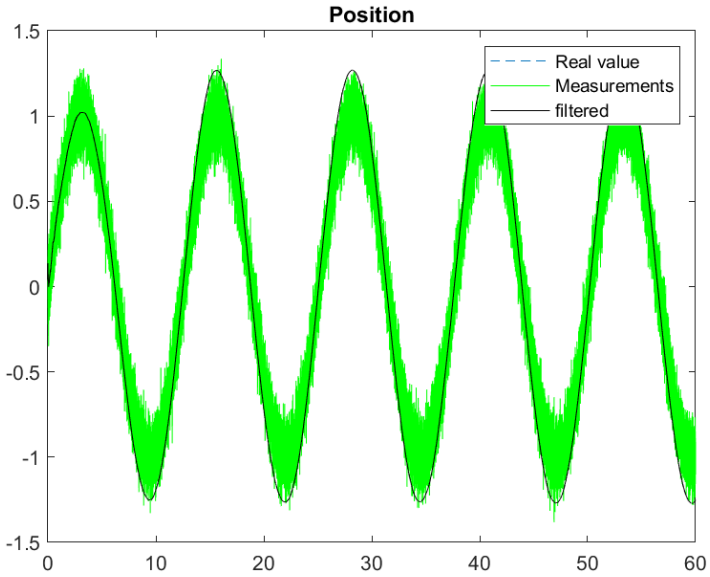
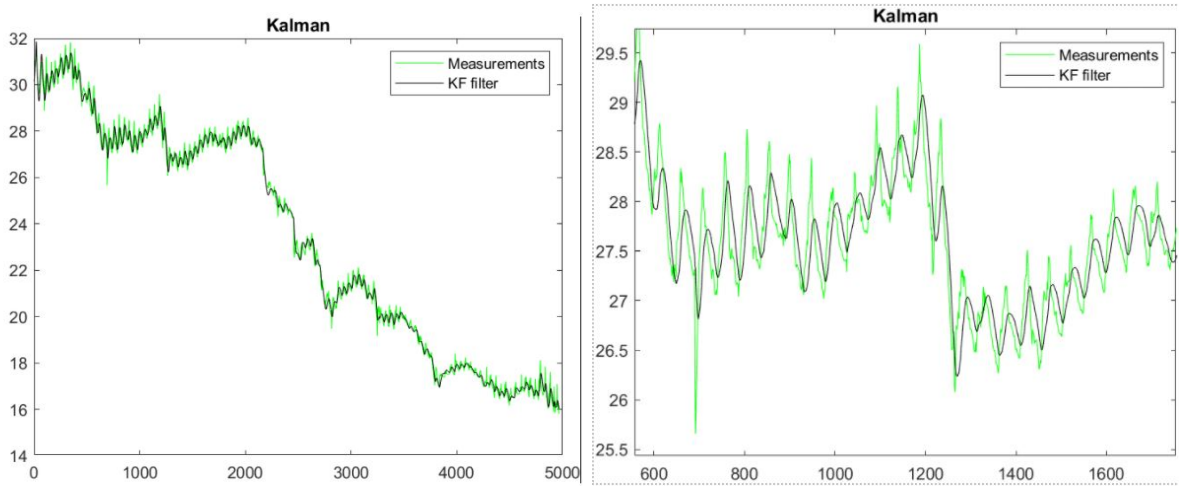


Figura 4.19: Sobre suavizado del filtrado por elección de σ_q muy pequeño.

4.4.1. Aproximación σ_q

Para las pruebas de la σ_q variable se utilizaron datos reales obtenidas de la herramienta IDbox para así poder recrear el funcionamiento que tendrán una vez implementadas.

A continuación se muestra el funcionamiento del algoritmo de la σ_q variable donde se ha cogido $n = 48$ puntos para los ajustes y $R = 0{,}5^2$



(a) Filtrado de la señal (b) Zoom del filtrado

Figura 4.20: Ejemplo de aplicación del algoritmo de σ_q variable sobre datos de la temperatura de una iglesia, en verde los valores medidos y en negro los filtrados. A la derecha se muestra el filtrado ampliado

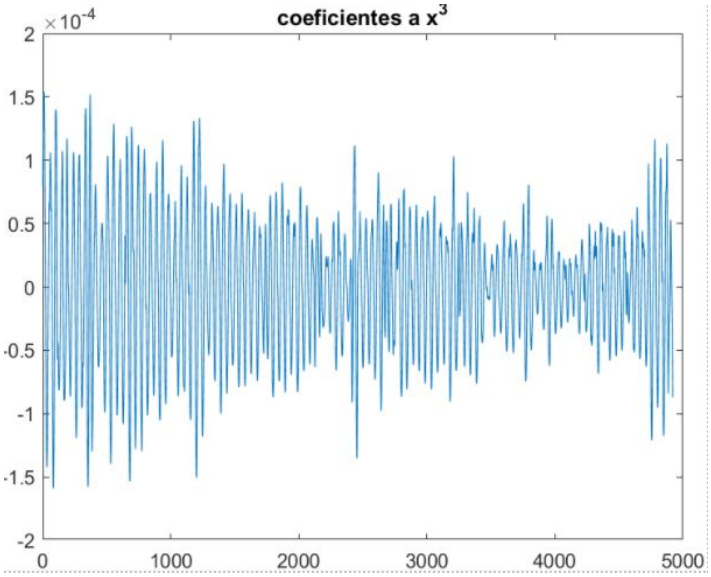
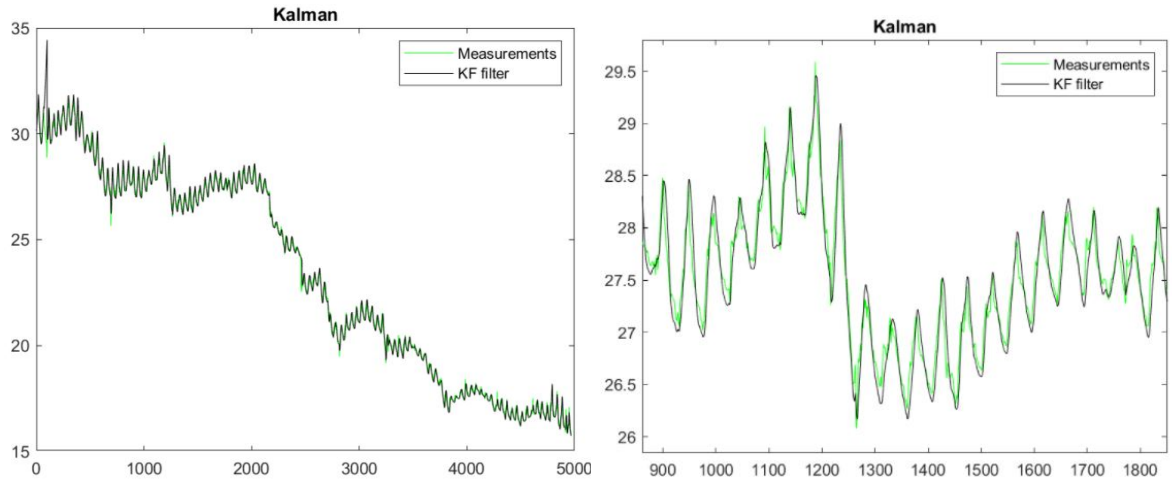


Figura 4.21: Valores de los coeficientes de orden 3 durante el filtrado de la señal 4.20

En este caso con el número de puntos escogido se puede observar en la figura 4.20 que se reduce el error pero puede llegar a perderse información de los picos. La

figura 4.23 representa los valores estimados del coeficiente cúbico del desarrollo en serie, como vemos toma valores con gran variabilidad (por lo que la premisa del modelo de kalman de que son aleatorios es apropiada). La dispersión nos da una idea del orden de magnitud de σ_q, pero además nuestro algoritmo puede ajustar este valor en períodos donde esta magnitud cambia.

Por otro lado, como se comentó el valor de n puede ser variable dependiendo de lo que busque el usuario. En la figura 4.22 se tomó $n = 12$



(a) Filtrado de la señal (b) Zoom del filtrado

Figura 4.22: Ejemplo de aplicación del algoritmo de σ_q variable sobre datos de la temperatura de una iglesia, en verde los valores medidos y en negro los filtrados. A la derecha se muestra el filtrado ampliado

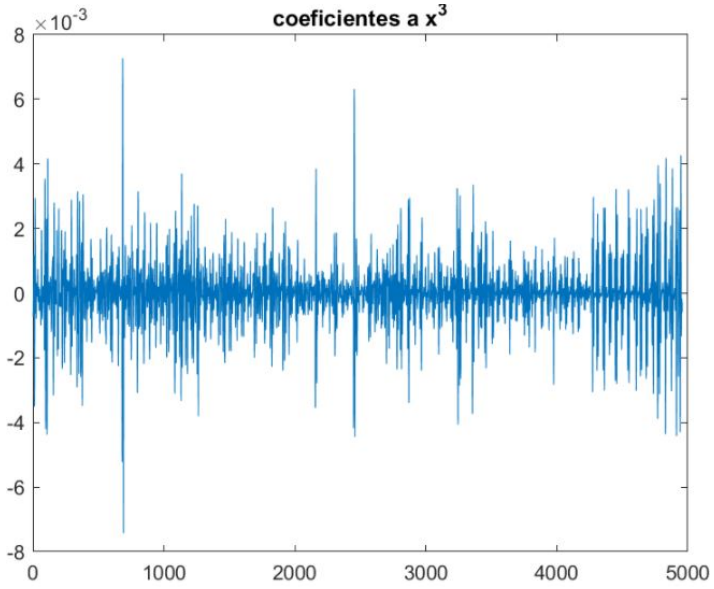


Figura 4.23: Valores de los coeficientes de orden 3 durante el filtrado de la señal **??**

Se observa claramente que la reducción del número de puntos hace que el filtro siga más a la señal.

Como en la herramienta IDbox se tenían diferentes señales a continuación se muestra el filtrado de una señal que no tenía nada que ver con la anterior para ver que el filtro es bastante robusto sea cual sea la señal de entrada. En la siguiente aplicación se tomó $n = 20$ y $R = 0{,}25^2$.

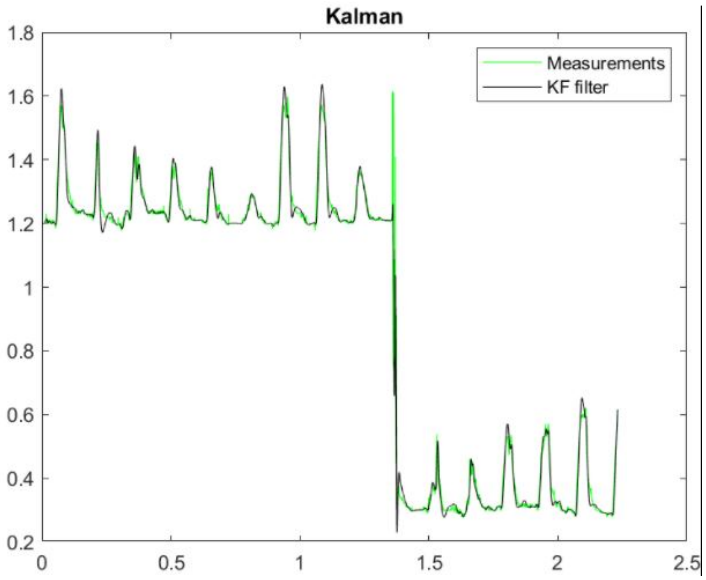


Figura 4.24: Ejemplo de aplicación del algoritmo de σ_q variable sobre datos de la concentración de ácidos en un tanque de agua, en verde los valores medidos y en negro los filtrados.

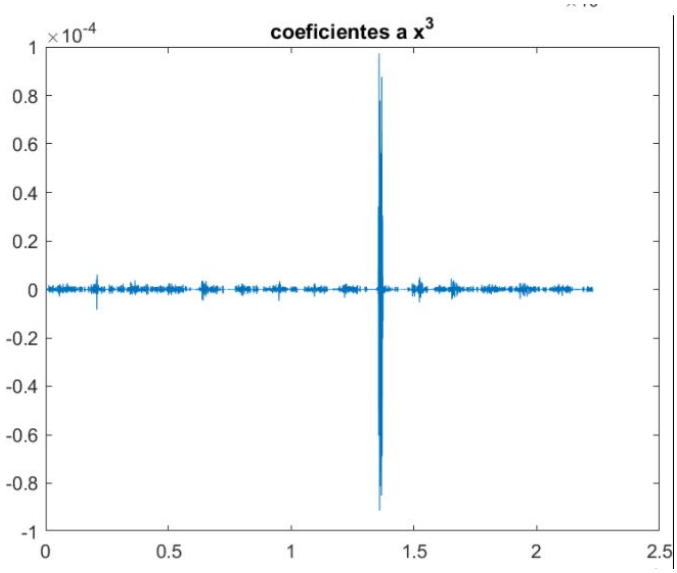


Figura 4.25: Valores de los coeficientes de orden 3 durante el filtrado de la señal 4.24

Se puede observar en la Figura 4.24 que el filtro consigue seguir muy bien a la señal, destacando que aun teniendo un salto abrupto consigue un filtrado consistente. Aun así, ampliando la imagen podemos observar en la figura 4.26 que el filtro sigue muy bien a la señal pero hay puntos donde sobre suaviza, esto se podría evitar disminuyendo n a cambio de una menor reducción del error.

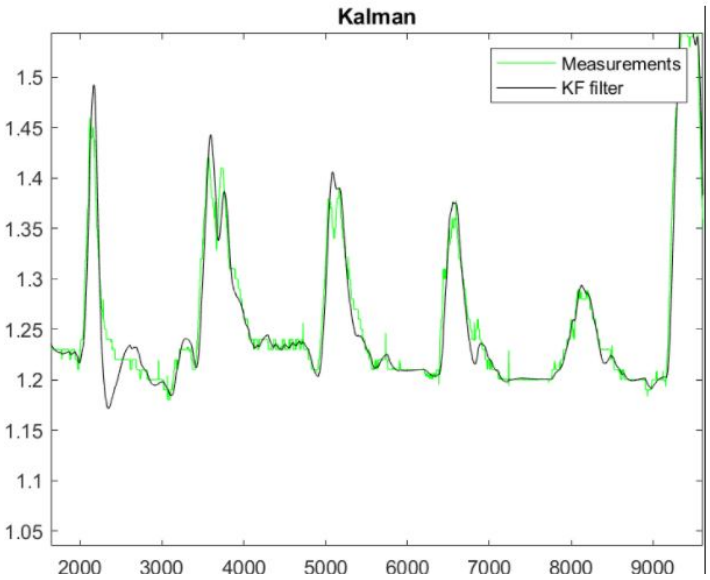


Figura 4.26: Figura 4.24 Ampliada

Todos los datos que se muestran en las anteriores pruebas de aplicación se obtuvieron de la herramienta IDbox. Para ello, se elegía en la herramienta la señal deseada y se descargaban los datos durante el periodo de tiempo deseado, una vez descargados los datos se cargaron en MATLAB. El código utilizado sobre estas señales es el 7.6 mostrado en el anexo

4.5. Ejemplo de uso en IDbox

A la hora de utilizar el filtro de Kalman en la herramienta IDbox, al usuario se le pedirá que introduzca dos valores

- Error de la medida o del aparato de medición de la señal R
- Número de puntos para ajustar n (En caso de no saber nada por defecto un valor pequeño (10) para evitar sobre suavizar o fallos del filtro.)

Ahora si el usuario está utilizando la función del filtro sobre un intervalo fijo de datos a parte del filtrado de Kalman tendrá la oportunidad de aplicar el suavizador RTS. En la figura 4.27 se puede ver como vería la señal en la herramienta simulado en MATLAB

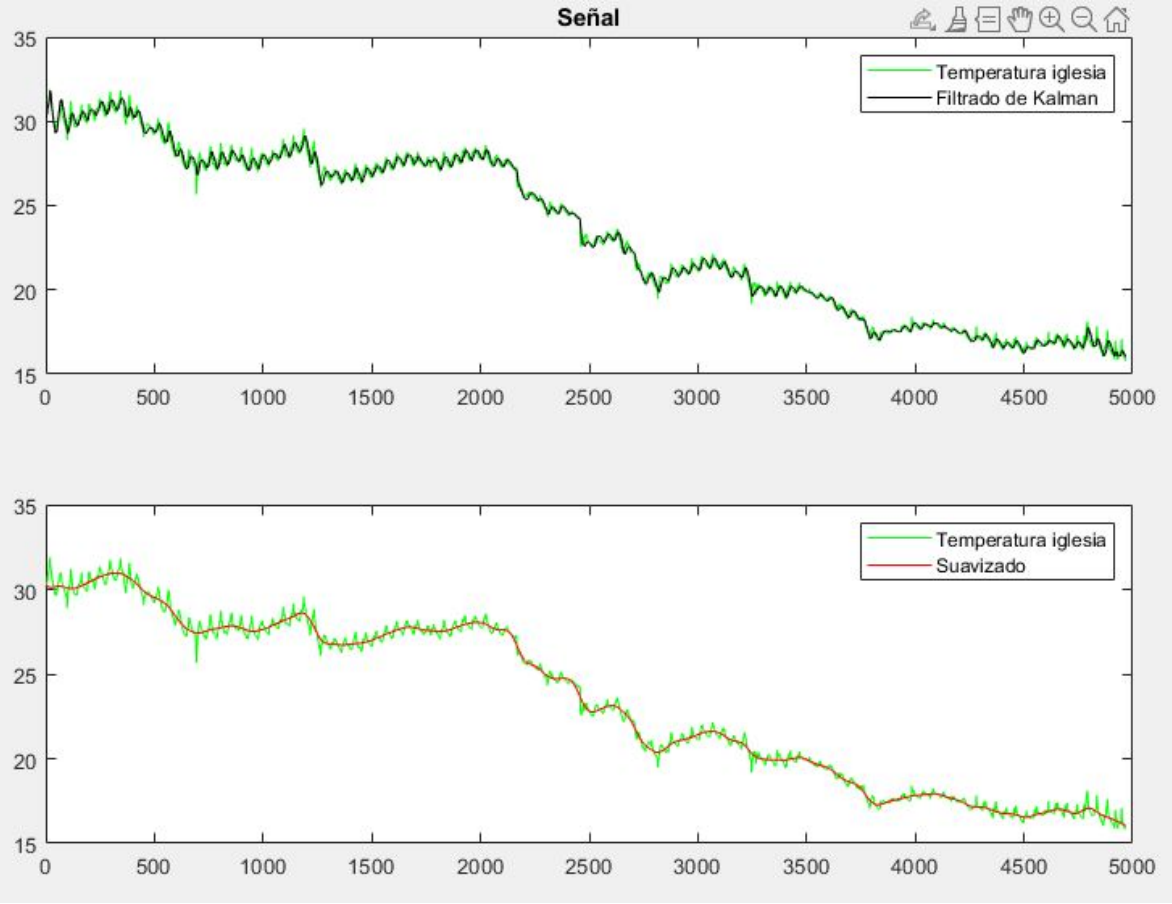


Figura 4.27: Ejemplo de como observaria el usuario la señal al aplicar el filtro de Kalman sobre su serie temporal

Se puede observar que el usuario tiene a disposición el filtro de Kalman para filtra y además la función RTS si lo que busca es suavizar la señal y quitar los picos de alta frecuencia que se puedan dar en la señal.

5. Implementación en IDbox

Una vez se tuvo el algoritmo de Kalman bien implementado con la σ_q variable, realizar las pruebas sobre diferentes tipos de datos que se tenían disponibles en IDbox y tras comentar el problema o mejor dicho la delicadeza de elegir el número de puntos para hacer el ajuste de σ_q con el C.I.C. se optó por dejar a elección del usuario el parámetro n, debido a que es quien mejor conoce la señal y sabe que filtrado desea y que información está dispuesto a perder al sobre suavizar. Además, en la herramienta IDbox la entrada de datos es demasiado variada como para dejar estos parámetros fijos ya que estamos hablando de señales que van desde la temperatura interior o exterior de un edificio hasta la concentración de ácidos en un tanque de agua de una empresa de neumáticos. En definitiva, situaciones muy complejas en las que la mejor elección la hará el usuario del filtro.

5.1. Algoritmo en C#

Por último, se implementó el algoritmo en el lenguaje de programación de IDbox que es C#. El proceso de migración fue bastante lento debido al cambio a un lenguaje de programación nuevo. Además, al principio no se sabía bien como cómo realizar operaciones matemáticas sobre las matrices en C#, pero este problema se solvento con la utilización del paquete NuGet de MATHNET.Numerics. ofreciendo funciones específicas para operar con matrices. Mientras se migraba el código se llegó a la conclusión de que el algoritmo se podía implementar para dos usos diferentes en IDbox.

Para la primera implementación en IDbox se decidió implementar el filtro como función de procesado de señales, es decir, el filtro trabaja con trenes de datos definidos y no a tiempo real. En este caso, el usuario coge la señal que quiera (una señal de una longitud N) y le puede aplicar el filtro en segundo plano. Además, se decidió que para esta parte también se le implementaría el suavizador RTS, ya que eran intervalos fijos y permitía su uso. Esta implementación tiene como finalidad otorgar al usuario un filtrado igual al que se observaba en la figura 4.27.

Y como última fase se implementó el filtro a tiempo real, haciendo que cada vez que entrara un nuevo dato en el sistema éste fuese procesado y devolviera al momento el valor estimado con los valores necesarios para seguir el ciclo del algoritmo.

En ambos casos, a nivel de usuario lo único que se pide para utilizar el filtro es

- Conocer una estimación del error de la medida R
- Saber bien la frecuencia de muestreo para coger el valor n para suavizar más o menos la señal

Una vez se tuvieron los dos códigos preparados y listos para su uso se dejaron a manos de otro equipo de C.I.C. para que llevaran a cabo el diseño de la interfaz gráfica y la implementación final en la herramienta.

6. Conclusiones

En este trabajo se ha abordado el problema de filtrado de series temporales realistas de sistemas de monitorización registrados por la empresa C.I.C Consulting. Para ello, se han estudiado diversas opciones de filtros, optando por el filtro de Kalman como la mejor solución debido a sus características de reducción del error como su recursividad.

Antes de implementarlo en la herramienta IDbox de la empresa, se han ido desarrollado programas en MATLAB para estudiar en detalle el rendimiento de diferentes versiones del algoritmo. Se ha podido comprobar que el filtro de Kalman es un estimador lineal óptimo que reduce el error cuadrático medio de la señal siempre y cuando se conozca el modelo dinámico de la señal que se filtra. Pero, debido a la falta de información de las señales que se tenía, se extendió el método habitual del Kalman. En primer lugar, utilizando un sistema dinámico teniendo en cuenta la primera derivada, y después se desarrolló en segundo orden de Taylor, a dos dimensiones.

Se ha añadido un sistema de filtrado basado en un validation gate, que se comprobó daba mucha más robustez al resultado ante fallos como valores anómalos o perdidas de señal. Se comprobó, que este filtrado combinado con el método 2D, daba una buena respuesta incluso en situaciones donde había pérdidas de señal prolongadas.

También, se ha visto la importancia que tiene a la hora de implementar el filtro de Kalman el conocer bien la fuente del error de la señal. Se han estudiado distintas formas funcionales para su aplicación. Por un lado se tenía que la forma de la matriz Q era delicada en cuanto al desempeño del filtrado, de modo que, se propuso una matriz definida a partir del término cúbico del desarrollo de Taylor, comprobándose que responde bien ante el modelo utilizado. Se comprobó que lo más crucial para obtener un filtrado óptimo es conocer el orden de magnitud de σ_q. Por ello, se ha desarrollado un algoritmo en el que se calcula de los mismos datos este parámetro σ_q, que básicamente representa el orden de magnitud del término de la derivada que se considera aleatorio, comprobándose que aumenta tremendamente el rendimiento del filtro en casos en que este orden de magnitud es desconocido.

Finalmente la implementación tanto en MATLAB como en IDbox, muestra unos resultados muy satisfactorios tanto sobre ejemplos simulados como sobre los datos reales.

7. Anexo

7.1. Códigos MATLAB

7.1.1. Algoritmo de Kalman

Función de predección:

```
function [x,P] = kf_predict(x,P,A,Q)

  %
  % Si no hay suficientes argumentos de entrada inicializar↩
      a 0
  %
  if nargin < 3
    A = [];
  end
  if nargin < 4
    Q = [];
  end
  %
  % Rellenamos con 0
  %
  if isempty(A)
    A = eye(size(x,1));
  end
  if isempty(Q)
    Q = zeros(size(x,1));
  end
  %
  %prediccion
  %
    x = A * x;
    P = A * P * A' + Q;
    \label{aaaaa}
```

Extracto de código 7.1: Código de la función de predicción del filtro

Función de actualización:

```
function [X,P,K,IM,IS] = kf_update(X,P,y,H,R,chi2)

  %
```

```
  % Si no tiene suficientes argumentos de entrada
  %
  if nargin < 5
    error('Too few arguments');
  end
  if nargin<6
      chi2=9;%por defecto coger 3 sigmas para el validation↩
          gate
  end
  %
  % update
  %
  IM = H*X;
  IS = (R + H*P*H');%Matriz de covarianza
  v=y-IM;
  if v'/IS*v <chi2 %-------------Validation gate↩
     --------------
                    %intervalo de sigma = 1, 2*sigma=4, 3*↩
                       sigma=9,.....
   K = P*H'/IS;
   X = X + K * (y-IM);
   P = P - K*IS*K';
  else %Si no pasa por la puerta no tomamos la medida y ↩
     segumos con la anterior predicci n
      X=X;
      P=P;
  end
  end
```

Extracto de código 7.2: Código de la función de corrección del filtro

7.1.2. Tracking Tiro parabólico

```
clc
clear
close all
N=1000;%Numero depuntos
dt=0.001;%tiempo de sampleo
t=dt*(1:N);%Vector del tiempo
F=[1 dt %Matriz de transici n
   0 1];
G=[-1/2*dt^2 %tiro parabolico
    -dt];
H=[1 0];
Q=[0 0
   0 0];%asumimos que no hay ruido
```

```
g=9.8;
I=eye(2);
%deinicion de las posiciones y velocidades iniciales
y0=100;
v0=0;
%inicializamos  el vector de estado
xt=zeros(2,N);
xt(:,1)=[y0;v0];
%generamos los valores del vector de estado(tiro parabolico↩
   )
for k=2:N
    xt(:,k)=F*xt(:,k-1)+G*g;
end
%generamos el ruido
R=4;%varianza del error en la posoicion
v=sqrt(R)*randn(1,N);
z=H*xt+v;
x=zeros(2,N);
x(:,1)=[10 ;0];
P=[50 0 %error en la posici n
    0 0.01];%error en la velocidad
for k=2:N
    %prediccion del vector de estado
    x(:,k)=F*x(:,k-1)+G*g;
    %Prediccion de la matriz de covarianza
    P=F*P*F'+Q;
    %Calculo de lamatriz de la ganancia de Kalman
    K=P*H'/(H*P*H'+R);
    %Actualizar/corregir el vector de estado
    x(:,k)=x(:,k)+K*(z(k)-H*x(:,k));
    %Actualizar la matriz de covarianza
    P=(I-K*H)*P;
end
figure(1)
subplot(211)
plot(t,z,'b-',t,x(1,:),'r--')
hold on
plot(t,xt(1,:),'k:','Linewidth',1.5)
legend('Medido','Estimado','verdadero')
axis([0 length(N) 90 110])
xlabel('t / s'),ylabel('Altura / m')
subplot(212)
plot(t,x(2,:),'Linewidth',2)
hold on
plot(t,xt(2,:),'r:','Linewidth',1.5)
legend('Estimado','verdadero')
```

```
xlabel('t / s'),ylabel('V / m s^{-1}')
```

Extracto de código 7.3: Código Matlab filtrado del tiro parabólico

7.1.3. Modelo 1D

```
  clc
  clear
  close all
  sd = .500;%ruido gaussiano con 0.5 de desviaci n
  dt = 0.01;
  w = .5;%frecuencia del seno
  T = (0:dt:60);
  %% --------------DIFERENTES SE ALES↩
     ----------------------------
  X=sin(w*T);
 %X=exp(-0.05*T).* sin(w*T);
 %X=polyval([.002 .093 .03],T)
 %↩
    ------------------------------------------------------------------↩

  Y = X + sd*randn(size(X));%A adimos el ruido a la se al
  %% ----------A ADIMOS MANUALMENTE OUTLIERS↩
     ----------------------
  %------------------OUTLIERS↩
     ---------------------------------------
%     Y(100)=5;Y(2000:2070)=6;Y(1000:1010)=4;Y(1,4450:4000)↩
   =-10;
% Y(1,4450:4700)=-10; a
  %↩
     ------------------------------------------------------------------↩


  %
  % Iniciamos el filtro
  %
  M = [0;0];
  P = diag([1 1]);
  R = sd^2;
  H = [1 0];%Solo medimos posici n
  q = 0.025;
  A=[1 dt;
      0 1];
  Q=[dt^4/4 dt^3/2;
      dt^3/2 dt^2];
  %
  % Guardamos y pintamos
```

```
  %
  MM = zeros(size(M,1),size(Y,2));
  PP = zeros(size(M,1),size(M,1),size(Y,2));

  for k=1:size(Y,2)
    %
    % Track with KF
    %
    [M,P] = kf_predict(M,P,A,Q);
    [M,P] = kf_update(M,P,Y(k),H,R,9);
    MM(:,k) = M;
    PP(:,:,k) = P;
  end
  figure(1)
  plot(T,Y(1,:),'r',T,MM(1,:),'k')
  legend('Measurements','Filtered estimate')
  %%
%RESIDUO
figure(2)
residuoKF=X(1,:)-MM(1,:);
histogram(residuoKF,50)
legend
clc
fprintf('Error sin filtro= %.5f\nError KF = %.5f\n',...
    sqrt(mean((X(1,100:end)-Y(1,100:end)).^2)),...%Raiz de ↩
       la media del residuo al cuadrado
    sqrt(mean((X(1,100:end)-MM(1,100:end)).^2)));
%% MAD
MAD=mad(residuoKF,1);
desv=sqrt(mean((X(1,100:end)-MM(1,100:end)).^2));
varianza=var(residuoKF);
clc
fprintf('Desviaci n KF= %.5f\n',...
    sqrt(varianza));
porc='%';
fprintf('MAD KF= %.5f\n',...
    MAD);
k=0;
rq=1;
for i=1:length(residuoKF)
   if sqrt(residuoKF(i)^2)>2*MAD%Equivalente a 1.5 sigma ↩
      aprox
       k=k+1;
   else
       mad2residuos(rq)=residuoKF(i);
       rq=rq+1;
   end
```

```
end
porcentaje=k/length(residuoKF) *100;
fprintf('El porcentaje de puntos que se desvian un MAD es ↩
   de = %.3f %%\n',...
     porcentaje);
%% %RESIDUO con MAD
figure(3)
histogram(residuoKF)
hold on
histogram( mad2residuos)
legend
```

Extracto de código 7.4: Código de Matlab de la implementación del modelo 1D

7.1.4. Modelo 2D

```
clc
clear
close all
% Stepsize
dt =0.001;
sd=0.1;%Standar deviation
% Process noise variance
q =1/dt^2;%input('Introduce valor de \sigma^2 :');
% Discretization of the continous-time system.
  A= [1 dt dt^2/2;
      0 1 dt;
      0 0 1];
  Q=[dt^6/36 dt^5/12 dt^4/6;
      dt^5/12 dt^4/4 dt^3/2;
      dt^4/6 dt^3/2 dt^2]*q;
      %   Q=[0 0 0;
%       0 0 0;
%       0 0 (1.2e-7)^2];
  q2=4.5e-7/dt^2;
  Q2=[dt^4/4 dt^3/2 dt^2/2;
       dt^3/2  dt^2 dt;
         dt^2/2 dt 1]*q2^2;
H = [1 0 0;
    0 0 0;
    0 0 0];
r=0.05;
R = diag([r r r]);%error de la medida
x = sawtooth(T);%
n=T;
```

```
z=x+sd*randn(size(x));
X(1,:)= x;
Y(1,:)=z;
%↩
   ------------------------------------------------------------------------↩

% Initial guesses for the state mean and covariance.
m = [Y(1,1) 0 0]';
P = diag([1 1 1]);
```

Extracto de código 7.5: Código de Matlab de la implementación del modelo 2D

7.1.5. Q variable

```
clc
clear
close all
load datosBridge1;
load datosTemp;
load DatosPrueba;
temp1=signal';
temp2=signal2';
%%
%Elegir datos
sd=0.52;
x=seno;
Y=x+sd*randn(size(x));
%%
%Y=temp1;
np=input('Introduce numero de puntos :');
%  np=1000;
%Vector de tiempo
T =0:length(Y)-1;
dt =1;
% Process noise variance
q =1e-3/(dt^3);%input('Introduce valor de \sigma^2 :');
%Error de la medida
mr=0.52;
% Discretization of the continous-time system.
  A= [1 dt dt^2/2;
      0 1 dt;
      0 0 1];

  Q=[dt^6/36 dt^5/12 dt^4/6;
      dt^5/12 dt^4/4 dt^3/2;
      dt^4/6 dt^3/2 dt^2]*q^2;
```

```
  %%
  %Otra matriz Q =F*q*transp(F)
% q2=0.000001/dt^2;
%      Q2=[dt^4/4 dt^3/2 dt^2/2;
%       dt^3/2  dt^2 dt;
%       dt^2/2 dt 1]*q2^2;

H = [1 0 0];
R = mr^2;
m = [Y(1,1) 0 0]';
P = diag([1 1 1]);
MM = zeros(size(m,1), size(Y,2));
PP = zeros(size(m,1), size(m,1), size(Y,2));
j=1;
for i = 1:size(Y,2)



   [m,P] = kf_predict(m,P,A,Q);
   [m,P] = kf_update(m,P,Y(:,i),H,R,10);
   MM(:,i) = m;
   PP(:,:,i) = P;
   if rem(i,np)==1
        if i>1
            p=polyfit(-np/2:np/2,Y(1,i-np:i),3);%Mirar bien↩
                el intervalo, sobre suaviza con muchos ↩
               puntos
            ax3(j)=p(1,1);
            j=j+1;
            if length(ax3)>10
                q=(mean(ax3(end-10:end).^2));%Cuadrado de ↩
                   la media ultimos 10 punos
                Q=[dt^6/36 dt^5/12 dt^4/6;
                    dt^5/12 dt^4/4 dt^3/2;
                    dt^4/6 dt^3/2 dt^2]*q;
            end
        end
   end
end

%%
figure(2)
plot(T,Y(1,:),'g',T,MM(1,:),'k')
title('Kalman');
 legend('Measurements','KF filter');
%%
```

```
figure(6)
plot(ax3)
title('coeficientes a x^3');
residuoKF=x(1,:)-MM(1,:);
%% MAD
MAD=mad(residuoKF,1);
desv=sqrt(mean((x(1,:)-MM(1,:)).^2));
varianza=var(residuoKF);
clc
fprintf('Desviaci n KF= %.5f\n',...
    sqrt(varianza));
porc='%';
fprintf('MAD KF= %.5f\n',...
    MAD);
k=0;
rq=1;
for i=1:length(residuoKF)
   if sqrt(residuoKF(i)^2)>4*MAD%Equivalente a 3 sigma ↩
      aproxx
       k=k+1;
   else
       mad2residuos(rq)=residuoKF(i);
       rq=rq+1;
   end
end
porcentaje=k/length(residuoKF) *100;
fprintf('El porcentaje de puntos que se desvian 4 MAD es de↩
    = %.3f %%\n',...
     porcentaje);

%% %RESIDUO
figure(6)
histogram(residuoKF)
hold on
histogram( mad2residuos)
legend
```

Extracto de código 7.6: Código modelo 2D con el algoritmo de σ_q

7.2. Función RTS

```
function [M,P,D] = rts(M,P,A,Q)

  if nargin < 4
```

```
  error('Too few arguments');
end

%
% extendemos las matrices
%
if size(A,3)==1
  A = repmat(A,[1 1 size(M,2)]);
end
if size(Q,3)==1
  Q = repmat(Q,[1 1 size(M,2)]);
end

%
% Run the smoother
%
D = zeros(size(M,1),size(M,1),size(M,2));
for k=(size(M,2)-1):-1:1
  P_pred   = A(:,:,k) * P(:,:,k) * A(:,:,k)' + Q(:,:,k);
  D(:,:,k) = P(:,:,k) * A(:,:,k)' / P_pred;
  M(:,k)   = M(:,k) + D(:,:,k) * (M(:,k+1) - A(:,:,k) * M↩
     (:,k));
  P(:,:,k) = P(:,:,k) + D(:,:,k) * (P(:,:,k+1) - P_pred) ↩
     * D(:,:,k)';
end
```

Extracto de código 7.7: Función del suavizador de intervalo fijo RTS

8. Bibliografía

[1] colaboradores de Wikipedia". Business Process Model and Notation (2021). URL https://es.wikipedia.org/wiki/Business_Process_Model_and_Notation.

[2] Haykin, S. *Adaptive Filter Theory* (Pearson, 2014), 5 edn.

[3] Raga, M. C. M. Filtro de kalman y sus aplicaciones (Universitat de Barcelona, 2018). URL https://doi.org/10.5772/intechopen.71731.

[4] Kalman filter - Wikipedia. URL https://es.xcv.wiki/wiki/Kalman_filter.

[5] Saho, K. Kalman filter for moving object tracking: Performance analysis and filter design. In de Oliveira Serra, G. L. (ed.) *Kalman Filters*, chap. 12 (IntechOpen, Rijeka, 2018). URL https://doi.org/10.5772/intechopen.71731.

[6] Williams, J. W. A study of second and third order models for the tracking subsystem of a radar guided missile (1988). URL https://calhoun.nps.edu/handle/10945/23127.

[7] Shu, H., Simon, E. P. & Ros, L. Third-order kalman filter: Tuning and steady-state performance. *IEEE Signal Processing Letters* **20**, 1082–1085 (2013).